**一、混淆简介**

**混淆的概念**：将Android项目进行打包之时，可以将项目里的包名、类名、变量名进行更改，使得代码不容易泄露,类似于对其apk中的文件加密.

**即：**在应用发布前，就需要对代码进行混淆处理，从而让我们代码即使被反编译，也难以阅读。

**混淆的作用**：

* 1.增加Apk反编译之后代码泄露的困难性
* 2.生成的apk体积会缩小

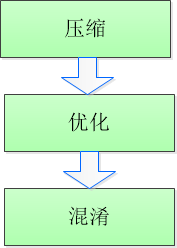
**1. 混淆流程**

Android中的“混淆”可以分为两部分:

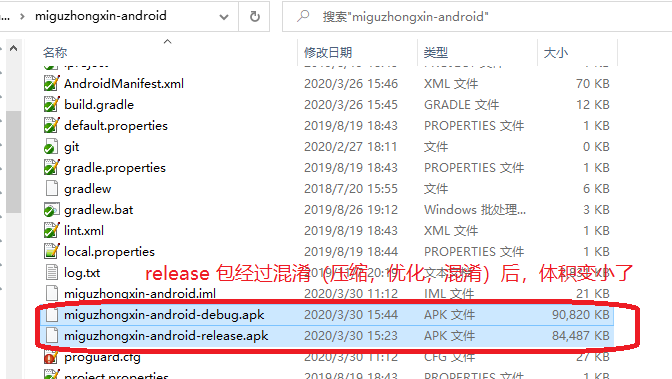
**一部分Java代码：**是Java 代码的优化与混淆，依靠 proguard混淆器来实现；

**一部分资源：**资源压缩是将移除项目及依赖的库中未被使用的资源(资源压缩严格意义上跟混淆没啥关系，但一般我们都会放一起用)。

代码混淆是包含了代码压缩、优化、混淆等一系列行为的过程。



**压缩：**移除无效的类、类成员、方法、属性等；



**优化：**分析和优化方法的二进制代码；根据proguard-android-optimize.txt中的描述，优化可能会造成一些潜在风险，不能保证在所有版本的Dalvik上都正常运行。

**混淆：**把类名、属性名、方法名替换为简短且无意义的名称；

**预校验：**添加预校验信息。这个预校验是作用在Java平台上的，Android平台上不需要这项功能，去掉之后还可以加快混淆速度。

这四个流程默认开启。

在Android项目中我们可以选择将“优化”和“预校验”关闭，对应命令是

-dontoptimize、 -dontpreverify（当然，默认的 proguard-android.txt文件已包含这两条混淆命令，不需要开发者额外配置）。

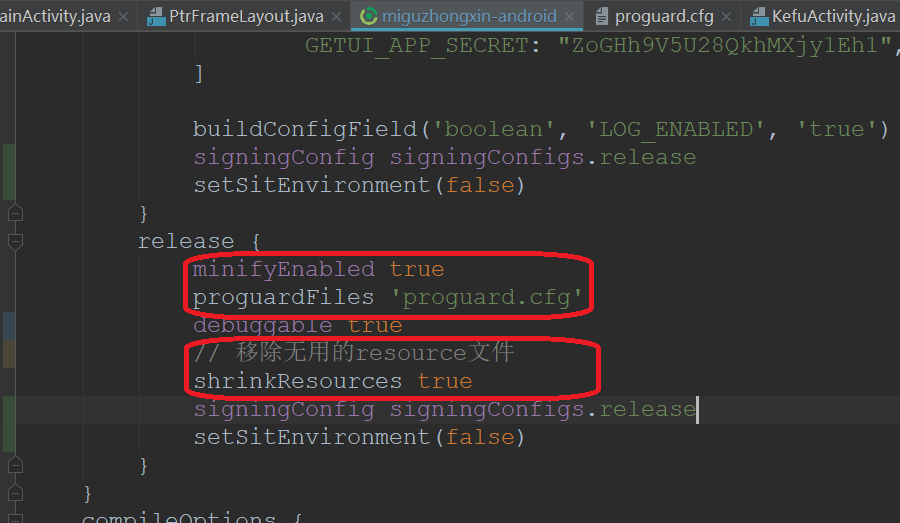
**2.工程（Project）中混淆配置**

因为开启混淆会使编译时间变长，所以debug模式下不开启。

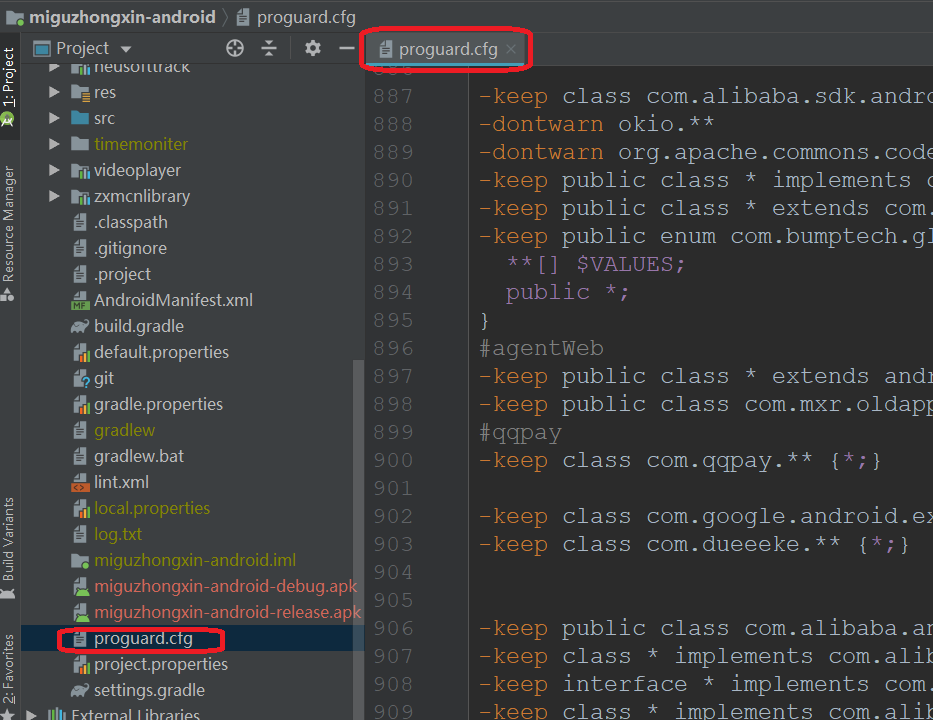
**我们需要做的是：**

* 将release下minifyEnabled的值改为true，打开混淆；
* 加上shrinkResources true，打开资源压缩。
* ProguardFiles配置混淆规则。如图中表示混淆规则文件为模块目录下proguard.cfg文件。

项目（Project）中混淆配置：



混淆规则文件（混淆白名单）



3，混淆规则文件（混淆白名单）

混淆规则文件为模块目录下proguard.cfg文件。

那么什么是混淆白名单呢？其实就是指定一些包名、类名、变量等不可以被混淆。假设没指定白名单就进行混淆打包，而某某类的类名被混淆了(假设变成了a)，那么可能其他引用或使用该类的类就找不到该类，说不定应用就会因此崩溃或是导致相应的功能无法使用。

**二、常见的混淆指令**

**1. 混淆流程3个指令：**

a.**压缩**（Shrinking）：

默认开启，用以减小应用体积，移除未被使用的类和成员；并且会在**优化**动作执行之后再次执行（因为优化后可能会再次暴露一些未被使用的类和成员）。

关闭压缩：

-dontshrink 关闭压缩

b.**优化**（Optimization）：

默认开启，在字节码级别执行优化，让应用运行的更快。

关闭优化：

-dontoptimize 关闭优化 -optimizationpasses n 表示proguard对代码进行迭代优化的次数，Android一般为5

c.**混淆**（Obfuscation）：

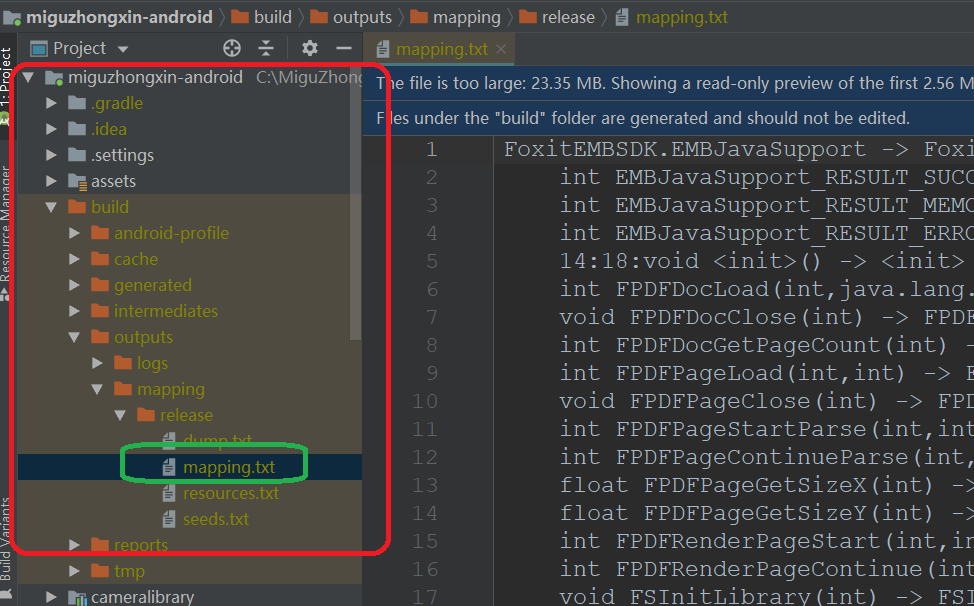
默认开启，增大反编译难度，类和类成员会被随机命名，除非用keep保护。

关闭混淆：

-dontobfuscate 关闭混淆

**特别说明：**

混淆后默认会在工程目录app/build/outputs/mapping/release下生成一个mapping.txt文件，这就是混淆规则，我们可以根据这个文件把混淆后的代码反推回源本的代码，所以这个文件很重要，注意保护好。



2.解出混淆文件

在 <sdk-root>/tools/proguard/路径下有附带的的反解工具（Window 系统为proguardgui.bat，Mac 或 Linux 系统为proguardgui.sh）。

这里以 Window 平台为例。双击运行 proguardgui.bat 后，可以看到左侧的一行菜单。点击 ReTrace，选择该混淆包对应的 mapping 文件（混淆后在 <module-name>/build/outputs/mapping/release/ 路径下会生成 mapping.txt 文件，它的作用是提供混淆前后类、方法、类成员等的对照表），再将 crash 的 stack trace 黏贴进输入框中，点击右下角的 ReTrace ，混淆后的堆栈信息就显示出来了。

**三、混淆——保持**

原则上，代码混淆后越乱越无规律越好，但有些地方我们是要避免混淆的，否则程序运行就会出错，所以就有了下面我们要教大家的，如何让自己的部分代码避免混淆从而防止出错。

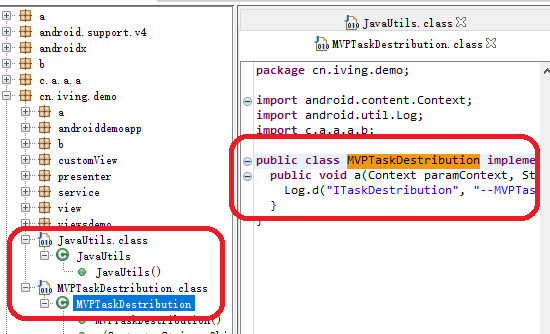
**3.1 保持元素不参与混淆的规则**

**SL:注意，混淆需要在主Module的规则文件中添加。**

**1.一颗星（\*）：**一颗星表示只是保持该包下的类名，而子包下的类名还是会被混淆。

示例：

-keep class cn.iving.demo.\*

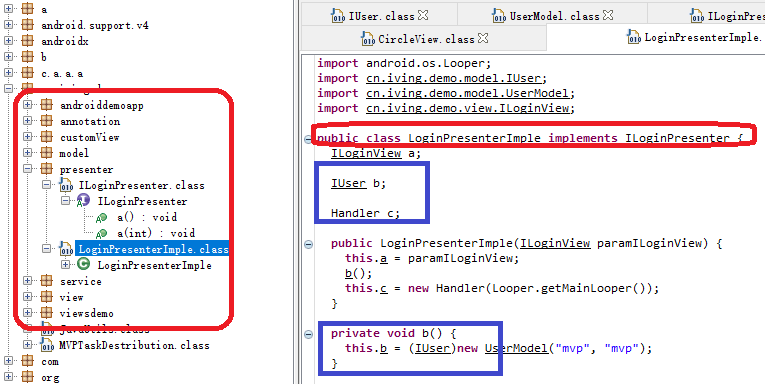


**2. 两颗星（\*\*）：**两颗星表示把本包和所含子包下的类名都保持,但类中的成员变量和成员方法名还是被混淆了。

示例：

-keep class cn.iving.demo.\*\*

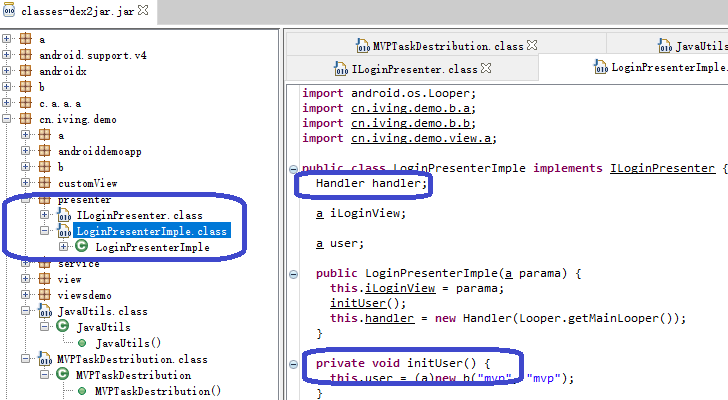
结果：



**3.类与方法均不混淆：**

如果既想保持类名，又想保持里面的内容不被混淆，怎么办？

-keep class cn.iving.demo.presenter.\*{\*;}



**4. 特殊语法**

我们也可以使用Java的基本规则来保护特定类不被混淆，比如我们可以用extend，implement等这些Java规则。如下例子就避免所有继承Activity的类被混淆

*-keep public class \* extends android.app.Activity*

要保留一个类中的内部类不被混淆则需要用$符号，如下例子表示保持ScriptFragment内部类JavaScriptInterface中的所有public内容不被混淆。

*-keepclassmembers class cc.ninty.chat.ui.fragment.ScriptFragment$JavaScriptInterface { public \*; }*

**四、混淆使用注意事项**

1，反射用到的类不混淆(否则反射可能出现问题)；

2.AndroidMainfest中的类不混淆，所以四大组件和Application的子类和Framework层下所有的类**默认**不会进行混淆（Android Studio中是无需加入的）。

3.自定义的View**默认**也不会被混淆（Android Studio中是无需加入的）。

4.有用到WebView的JS调用也需要保证写的接口方法不混淆，因为混淆后H5 的jS方法找不到；

5.jni方法不可混淆，因为这个方法需要和native方法保持一致；

6.与服务端交互时，使用GSON、fastjson等框架解析服务端数据时，所写的JSON对象类不混淆，否则无法将JSON解析成对应的对象；

7.使用第三方开源库或者引用其他第三方的SDK包时，如果有特别要求，也需要在混淆文件中加入对应的混淆规则；

8.Parcelable的子类和Creator静态成员变量不混淆，否则会产生Android.os.BadParcelableException异常；

-keep class \* implements Android.os.Parcelable { # 保持Parcelable不被混淆 public static final Android.os.Parcelable$Creator \*; }

9. 使用enum类型时需要注意避免以下两个方法混淆，因为enum类的特殊性，以下两个方法会被反射调用。

**附录：**

1.混淆官方文档：<https://www.guardsquare.com/en/products/proguard>。

2.参考资料：<https://www.jianshu.com/p/7436a1a32891>